Converting LaTeX to MathML: the Tralics algorithms

José Grimm

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Converting LaTeX to MathML: the Tralics algorithms

José Grimm

N° 6373

November 2007

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6373--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Converting LaTeX to MathML: the Tralics algorithms

José Grimm*

Thème NUM — Systèmes numériques
Équipe-Projet Apics

Rapport de recherche n° 6373 — November 2007 — 16 pages

Abstract: This paper describes how *Tralics* converts a sequence characters into a sequence of tokens, into a math list, and finally into a MathML formula. Tokenisation rules are the same as in TEX, the meaning of these tokens is the same as in LATEX, and can be given in packages. Math formulas are handled in the same spirit as TEX, but construction of the MathML result is not obvious, due to particularities of both TEX and MathML.

Key-words: Latex, XML, HTML, MathML, Tralics

* Email: Jose.Grimm@sophia.inria.fr

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Convertir du LaTeX en MathML : les algorithmes de Tralics

Résumé : Ce document explique comment *Tralics* convertit une formule de mathématique en objet MathML. Les mêmes règles que TEX sont appliquées pour convertir une suite de caractères en suite de lexèmes, ces lexèmes ont la même signification que dans LATEX, et peuvent être définies dans des paquetages. En mode mathématiques, ces lexèmes sont regroupés en listes, et traitées dans le même esprit que TEX, les règles précises, décrites dans ce document, différant considérablement à cause de certaines particularités de TEX et de MathML.

Mots-clés : Latex, XML, HTML, MathML, Tralics

1 Introduction

Tralics is a LaTeX to XML translator, described in [4] and [5]. It is used by Inria for production of its Annual Activity Report in HTML and Pdf. All math formulas found in the HTML pages are images obtained by converting pieces of the XML document via TeX and tools designed by D. Carlisle and S. Rahtz, see [2]. The math formulas conform to the MathML DTD [3] and can be inserted directly in a HTML document; some browsers interpret them natively, others require a plug-in. The header of the HTML file may be

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
 "http://www.w3.org/TR/MathML2/dtd/xhtml-math11-f.dtd">
```

Take for instance the following equation:

$$\int_0^\infty f(x+y)dx = |z| \ . \tag{1}$$

translated by *Tralics* as

```
<math mode='display' xmlns='http://www.w3.org/1998/Math/MathML'>
  <mrow>
   <msubsup><mo>&#x0222B;</mo><mn>0</mn> <mi>&#x0221E;</mi></msubsup>
   <mi>f</mi><mo>(</mo><mi>x</mi><mo>+</mo><mi>y</mi><mo>)</mo>
   <mi>d</mi><mi>x</mi><mo>=</mo>
   <mrow><mo>|</mo><mi>z</mi><mo>|</mo></mrow>
   <mspace width='3.33333pt'/><mo>.</mo>
  </mrow>
</math>
```

It happens that Amaya, one of the first browsers that renders MathML, shows very large parentheses, instead of normal ones. We shall explain in this document how *Tralics* converts input characters into MathML code, and the algorithm it uses in order to get a correct size for delimiters. A description of all math commands can be found in [6], its HTML version[1] contains examples of all constructs. By the way, the previous expression was entered as follows, and you can see how we forced *Tralics* to consider the parentheses as ordinary math objects.

```
1    \[\int _0^\infty f\mathord(x+y\mathord) dx = |z|~.\]
```

2 General Principles

In general, a compiler is formed of a lexer, a parser, a semantics analyser, a code generator. The purpose of the parser is to build a tree, using the tokens given by the lexer, representing the input; it is converted by the semantics analyser into an output tree, that is converted into a sequence of bytes by the code generator. The behaviour is TeX is special, in that tokens are evaluated as soon as possible, and no parse tree is created: the code generator receives lists of characters, boxes, glue and the like, typesets them (i.e., splits paragraphs into lines, fixes glue, adds page breaks, etc.) and converts the result into a sequence of bytes that are written in the DVI or Pdf file. In the case of a math expression, an intermediate tree is constructed and processed, and then converted into an ordinary list of characters, boxes, and the like. The main difference between *Tralics* and TeX is, of course, the code generator, but the handling of math expressions is, in principle, the same. There may be some subtle differences, as demonstrated by the following example, which will be explained later:

```
2    $\tracingall\sqrt\frac{1}{2}$
```

[1] Available on http://www-sop.inria.fr/apics/tralics

The lexer is the program that converts lines of text into tokens. There are two kinds of tokens in TEX, commands and characters. For instance `\[` and `\int` are two commands. In the case of a multi-letter control sequence, the space that follows is discarded. The meaning of the command is defined (in an internal table) by two integers: the command code and a subtype. For instance the command code of `\mathord` means: change the type of the item that follows in the current math list, and the subtype is the value of the new type (here Ord). In *Tralics* `\alpha` and `\beta` have the same command code meaning ordinary math symbol, and the subtype is the index of the associated XML object in a table[2]. In the case of a character, a pair of integers is created by the lexer; the command code is replaced by the category code of the character, and the subtype by the ASCII value of the character (in fact, *Tralics* is not restricted to 7-bit characters; instead of using the `\texteuro` command, you can use `^^^^20ac` or an input encoding such as latin9 that provides such a character). Letters have category code 11, most other characters have category code 12. In the example above, the dollar, underscore, and hat characters have a special meaning, they are short-cuts for math mode. The space character has a special category code; it is usually ignored in math mode. Finally the tilde is an active character (behaves like a command).

A primitive command is one defined in the Pascal (for TEX) or C++ source (for *Tralics*), as opposed to user-defined commands that are defined in a format file, a package, or in the document under translation. For instance, the tilde active character is a user defined command that takes no argument, and expands to `\nobreakspace` (a special rule of *Tralics* converts this into an ordinary math space in math mode). The `\frac` command takes two arguments (generally delimited by braces), its expansion is `{1\over2}`. The `\sqrt` command expands to some primitive that puts a square root around its argument. Because TEX uses a two-pass mechanism, the braces that delimit the argument of the square root are provided by the expansion of `\frac`. The `\over` command is a bit strange: its arguments are not enclosed in braces, but the braces define a scope. The `\over` command should not be used and *Tralics* handles `\frac` as a primitive. This means that the second example does not work.

The 'expand' procedure is one part of the translator, its purpose is to deal with user defined commands. It handles also conditionals. Some primitive commands are mode-independent, for instance the commands that modify internal quantities (category codes, equation numbers, etc). Remaining commands (including characters) add material to the dvi file (this can be done indirectly, you can put things in boxes, duplicate or ignore them). In the case of *Tralics*, an XML tree is produced instead of a dvi file, and this part of the program differs considerably from TEX. For instance, TEX splits paragraphs into line, but not *Tralics*, so that commands that control the line-breaking or page-breaking algorithm have a completely different implementation. Of the three modes of TEX, vertical, horizontal and math, the most particular one is math mode, and the purpose of this paper is to describe its implementation in *Tralics*.

3 Debug

The error message produced by *Tralics* for the example line 2 is the following.

```
3 Error signaled at line 1 of file tty:
4 Missing { inserted before unexpected }
```

This is because the expression was read as `\sqrt{\frac}` and `\frac` got a closing brace, instead of an opening one. Now, you are in trouble, because the inserted open brace does not match a closing brace[3]. In fact you will see this

[2] One could imagine a command code meaning: select a Greek letter in the current math font, this is a possible extension.

[3] Before version 2.10.9, the situation was worse, because *Tralics* removed the offending closing brace, and handled `\par` like ordinary non-math commands

```
Error signaled at line 1 of file tty:
Extra $ ignored while scanning argument of \sqrt.
```

You can see that you are in even greater trouble, because *Tralics* will never find the closing dollar sign if it continues like that. Notice however that *Tralics* explains here why the dollar sign is invalid: it expects the closing delimiter for the argument of the `\sqrt` command. Assume that we have two lines of text, followed by two empty lines.

```
Error signaled at line 4 of file tty:
Unexpected \par while scanning argument of \sqrt.
```

You cannot put end-of-paragraph commands in a math formula. The effect of such a mistake is radical: it stops parsing the expression. In our case, it provides a closing brace. The second empty line is also read as `\par`, this signals an error and provides the closing dollar sign.

Consider now the the example on line 1, without the `\mathord`. When you compile it in verbose mode, the transcript file will contain the following.

```
[1] \[\int _0^\infty f(x+y) dx = |z|~. \]
\[->$$
{math shift character}
+stack: level + 2 for math entered on line 1
~ ->\nobreakspace
\]->$$
+stack: level - 2 for math from line 1
```

The first line is printed when `\tracingoutput` is positive and a line is read from a file; lines containing a command followed by an arrow and some text are printed when `\tracingmacros` is positive and a command is expanded; lines that start with a plus sign are produced when `\tracingrestores` is positive, and the main stack (called semantic nest by Knuth) is modified, for instance when leaving a group restoration of variables is shown; lines that are enclosed in braces are printed when `\tracingcommands` is positive, and a command is evaluated: here it is the first dollar sign. Math commands like `\int` do not appear in the transcript file: outside math mode an error is signaled, in math mode these tokens are added to the math list, and the list is dumped as a whole, for instance

```
Math: $$\int_0^\infty f(x+y) dx = |z|~. $$
```

You can say `\tracingall`, in this case the transcript file contains everything that is needed in order to understand why *Tralics* produces strange results, including errors. When `\tracingmath` is true, the following lines will be printed by the fencing algorithm described in the last section of this paper.

```
MF: After find paren0 0b 2l 4B 6r 9R 10m 12m 15b
MF: sublist start=0 2l 6r 9R 10m 12m 15R
MF: Find paren2 k=9 2l 6r
MF: Find paren1 (1, 8) 2l 6r
MF: OK 2 6
MF: Find paren2 k=15 10m 12m
MF: Find paren1 (10, 14) 10m 12m
MF: BB 10 14
```

4 The Math Scanner

The first pass of the math translator produces a math list, this is a list of extended tokens; the additional field is generally the value of the current math font (one of the 15 fonts defined by MathML). The algorithm may replace a TEX token by an XML value, in this case the additional field indicates the role of the object (relation, binary, opening delimiter, etc; this is called the *kind*

in [7]). In the case of a command like `\frac` that takes arguments, each argument is defined by a separate math list, the additional field is a pointer to the argument list. Each math list has a type (if the list comes from an environment, the type encodes the name of the environment).

The scanner is defined by the following set of rules:

- The 'expand' procedure is called; in particular user defined commands are evaluated, as well as conditionals.
- Mode independent commands are evaluated as usual. Evaluation has no effect on the current math list.
- Non-math commands should not appear, an error will be signaled later.
- In general, the current token is added to the trace, this trace may be printed to the transcript file at the end.
- Special case of fonts. If the current token is `\rm`, `\textrm`, `\rmfamily`, `\mathrm`, a math font switch token N is constructed (in this case, it selects font number 1). If the current command takes no argument (like `\rm`), it will be replaced by N. Otherwise the argument is read, call it L; if the current font is O, then NLO is read again (see note 1 below).
- The effect of a math font is to change an internal variable that holds the current font value.
- Special case of a dollar when a tag is present. There is a hidden end-of-math hook in *Tralics*, used by the `\tag` command. If a dollar sign is seen and the hook is not empty, then the hook is re-inserted (as well as the dollar sign) and cleared.
- Tokens `\begingroup` or `\endgroup` can be used to increase or decrease nesting (these tokens act as group delimiters). The same effect can be achieved with braces, but these define a math sub-list. In the case of `\left` and `\right`, a delimiter is moreover scanned. See note 2.
- If the current token is `\begin` or `\end`, this is the start or end of an environment; it could be a user defined environment (handled like a user defined command), or a math environment (producing an array). It is parsed in the obvious way. Some math expressions are formed of a unique environment (so that the end-of-math hook must sometimes be inserted).
- Special case of `&`, `\\` or `\multicolumn`. These commands can appear only in a table, either in text mode or math mode. Special parsing rules are needed in math mode. Strange errors may be signaled.
- If the current token is a dollar sign, it is the start or the end of a new math formula. Action depends on the type of the formula to be created. If it's a simple math formula, the dollar indicates the end of the formula; if it's a display math formula, the next token (after expansion) is considered, it should be a second dollar sign; this indicates the end of the formula. If *Tralics* scans the content of a `\hbox` or friends, this indicates the start of a subformula. See Note 3.
- There are commands that take arguments, some are optional, some are mandatory; for instance, `\sqrt` takes an optional argument, `\genfrac` has a strange syntax. The parser returns an array of sublists, one per argument. See note 6.
- Commands like `\text`, `\mbox` and `\hbox` read an argument; in the case of `\hbox`, the `\everyhbox` token list is inserted; otherwise they behave the same. Other commands that construct boxes are forbidden (for instance `\vbox`, or extensions like `\xbox`).

- Glue is normalised. This is a bit tricky, but `\mskip18mu` is replaced by `\hspace{10pt}`; in general a glue value is read, transformed into a dimension, and handled like an ordinary command with an argument.
- Ordinary commands are added to the list, as well as special characters. The difference between `\texteuro` and `\alpha` is that the euro sign is character U+20AC, treated as an ordinary identifier in math mode, while the alpha letter is known (a bit later in the conversion process) to be an ordinary identifier, and `\cap` is a binary operator. The translation of a math symbol could be an entity name `α` or its value U+3B1. The current font is ignored, although Unicode has variants for Greek letters.
- Characters are added to the lists, together with the current font.

5 Notes

Note 1. A lot of work on math translation was done when T. Bouche showed interest in putting abstracts of journals like the Annales de l'Institut Fourier on the Web using MathML (see [1]). We provide the no-mathml mode, in which the translation directly matches the value of the math list. For instance, translation of formula 1 is

```
25 <texmath>\int _0^\infty f\mathord (x+y\mathord ) dx = |z|~.</texmath>
```

Normally, translation of

```
26  $12 ab \bf Cd \it Ef \cal Gh$
```

contains

```
27   <mn>12</mn>
28   <mi>a</mi><mi>b</mi>
29   <mi>&#x1D402;&#x1D41D;</mi>
30   <mi>&#x1D438;&#x1D453;</mi>
31   <mi>&Gscr;&hscr;</mi>
```

but it can be changed (via an integer variable) to

```
32   <mn>12</mn>
33   <mi>a</mi><mi>b</mi>
34   <mi mathvariant='bold'>Cd</mi>
35   <mi mathvariant='italic'>Ef</mi>
36   <mi mathvariant='script'>Gh</mi>
```

You can also ask *Tralics* to replace the quantity `𝒢` by `&#1D4A2;`. The essential difference is that a browser that has no font containing character U+1D402 will use a question sign or a black square in the first case, and a normal C, rather than a bold one, in the second case. When *Tralics* sees the bold C, it constructs an element formed of all characters that follow, provided that they have category letter, and are one of the 26 letters (upper case or lower case). Translation would be the same with a space after C, but other commands (for instance `\relax` or a font change) inhibits the mechanism. This mechanism is not applied in the case of the default font (lines 28 and 33).

Instead of `\bf`, you should use `\textbf` or `\mathbf`. These are commands that take arguments delimited by braces. These braces may be used by LaTeX as group delimiters. Hence *Tralics* inserts `\begingroup` and `\endgroup` around the argument (it is not clear why *Tralics* does not use braces).

Note 1bis. For the following input

```
37 $A \mathbf{B \mathit{C} D} E$
```

we get the following trace

```
$A \mml@font@bold\begingroup B \mml@font@italic
        \begingroup C\endgroup\mml@font@bold D\endgroup\mml@font@normal E$
```

and the math list (as well as the no-mathml output) is the same, without the grouping delimiter:

```
<texmath type='inline'>A \mml@font@bold B \mml@font@italic
C\mml@font@bold  D\mml@font@normal  E</texmath>
```

This looks a bit strange, but you can change the definition of either `\mathbf` or `\mml@font@bold` (let's denote this token by `\MFB` for simplicity). In note 4 below, we explain the simplest method: redefine `\mathbf` as a command that takes no argument and prints as `\mathbf`. In this case, the braces are considered as group delimiters, and interpretation of the formula can change if, by default, they are not considered as group delimiters. We can redefine `\MBF` so that it prints `\bf`. We must take care to insert a space, since otherwise you will see `\bfB`. The next difficulty is that a font switch token has to be inserted between C and D; in order for this to be bold, we need first that `\MBF` changes the value, and that the change induced by switching to italics is restored. This implies that we must take care when redefining `\MBF`, and *Tralics* must take care of grouping. Here is a solution

```
\makeatletter
\def\mml@font@bold{\@curmathfont=2\string\bf\space}
\def\mml@font@italic{\@curmathfont=3\string\it\space}
\def\mml@font@normal{\@curmathfont=0\string\normalfont\space}
$A \mathbf{B \mathit{C} D} E$
```

Translation is

```
<texmath type='inline'>A \bf B \it C\bf  D\normalfont  E</texmath>
```

Note 2. This example demonstrates grouping:

```
\def\foo{\A\def\A{0}\A}\def\A{1}
${\foo}={x\foo}=\left[\foo\right]=\begingroup\foo\endgroup\foo=\foo$
```

You should see $10 = x10 = [10] = 1010 = 00$. The MathML formula contains chunks separated by an equals sign; the first token is the integer 10; the second token is a `<mrow>` that contains x and 10 (the value of `\A` is restored, and math a list is converted into an `\mrow` then comes a `<mfenced>` element containing 10, it is followed by 1010, and then 00. The `\endgroup` token restores the value of `\A`, but does not create a sublist, and there is nothing that inhibits the conversion of the four digits into a number. After `\left` or `\right`, a delimiter is needed. This means that the next token is fully expanded, and `\relax` tokens are discarded, and one of the known delimiters must be found.

Note 3. In plain TEX, a math formula always starts and ends with a dollar sign. In LATEX, you can use the construction `\(...\)` for inline math, and `\[...\]` for display math; in these cases, we have commands that expand to dollar signs. LATEX provides also `math` and `displaymath` environments, but these are less used. There are more complex environments like `equation`, `align`, etc. Outside math mode, if a dollar sign is seen, math mode is entered. Next token is read, without expansion. If this token is a dollar (in fact, has category 3) display math will be entered. Otherwise, the token is re-read, unless it is `\relax`. In any case, when math mode is entered, the current mode will be normal math, or display math, and one of the token lists `\everymath` or `\everydisplay` will be inserted.

In the case of a display math formula, the equivalent of `\par` is inserted before and after the math formula (this has no consequence on the parsing). The result of the translation is a `<math>` element in a `<formula>` element; both elements hold an attribute that tells if the mode is inline or display. In display mode, a label is allowed, producing an attribute of the `<formula>`.

Note 4. The command `\ensuremath` takes an argument and evaluates it in math mode. It is coded like that: in math mode, the expansion is the argument, outside math the expansion is

dollar, argument, dollar. However, if the argument is empty, this is considered as a double dollar, hence starts display math. For this reason, a `\relax` token is added in front. As explained above, this token is discarded by *Tralics*. In a case like this

```
\everymath{\let\bf\relax}
$\bf x$
```

the `\bf` token is not discarded. Since its meaning is `\relax`, it will be ignored. Consider the following variant

```
\everymath{\let\textit\relax}
$ x= \textit{y} +z$
```

If you compile in no-mathml mode, these tokens, whose meaning is `\relax`, will show in the result, and you get:

```
<texmath type='inline'>\bf x</texmath>
<texmath type='inline'> x= \textit {y} +z</texmath>
```

Note 5. If you add braces around a sequence of tokens this produces a math list; after translation elements are enclosed in a `<mrow>` element. This produces a scope that controls the size of delimiters inside it. This element is not added if the list has a single element, except in some special cases: In a case like `$\displaystyle{\sum}'$`, the `<mrow>` is required if you want the prime to be on the right of the sum, rather than on the top. In the case of `${x_2}_3$`, an error occurs if there is no `<mrow>` and we try to convert the XML into Pdf. Note that TEX has a strange exception; if the math list is a single Acc atom, the atom itself is appended to the list. This means that, in a case like `${{\bar x}_2}_3$`, the outer braces are ignored (adding scripts does not changed the type) and a double subscript error is signaled; this behaviour is not implemented in *Tralics*.

Note 6. Here is the list of all functions that have a special syntax.

```
$\genfrac (){0pt}3{foo}{bar}$
$\sqrt{x}, \sqrt[x]{y}$
$\xleftarrow{u} \xleftarrow[v]{u}\xrightearrow{u} \xrightarrow[v]{u}$
$\smash{x} \smash[b]{x} \cfrac{x}{y} \cfrac[r]{x}{y}$
$\operatorname{sin}\operatorname*{cos}$
$\hat\relax y \sqrt\relax{x}$
$\mathchoice \relax\relax {1}\relax{2}{3}{4}$
% $x\relax_y\relax^z$
$\mathmi{x} \mathmi[a][b]{x} \mathbox[a][b]{foo}{x=y}$
```

The last line shows extensions provided by *Tralics*. If an odd number of optional arguments is given, the last one is ignored. In all these cases, the braces act as a group; optional `\relax` is ignored before the arguments of `\mathchoice`, `\root`, and commands that produce an accent. The line that is commented out is parsed normally, the `\relax` tokens are removed later on, when *Tralics* attaches the script to the kernel. These rules concerning `\relax` are strange, but implemented as in TEX (Start of Chapter 26 of [7] shows that a 'filler', defined in the middle of Chapter 24, is allowed before an open brace.)

6 Special Hacks

There is a special counter, associated to the nomathml option of the program. In this case, the translation of the math formula is the internal list converted into a string in straightforward manner. For instance

```
\newcommand\abs[1]{|#1|}
\[\int _0^\infty f(\frac xy) dx = \abs{z}~. \]
```

translates as

```
<texmath type='display'>\int _0^\infty f(\frac{x}{y}) dx = |z|~. </texmath>
```

In what follows, we shall assume that the counter is zero. There is a second counter, associated to the trivialmath option; it is described in details in the main *Tralics* documentation, and the purpose is to convert math formulas into non-math, in the following cases:

```
a single letter $x$, a single number $123$
a greek letter $\alpha$, a superscript or a subscript with text
only as $_{\bf foo}$, or numbers with special
exponents, for instance $2^{nd}$, $3^{i\grave{e}me}$.
```

This translates as

```
<p>a single letter
<formula type='inline'><simplemath>x</simplemath></formula>,
a single number 123
a greek letter &alpha;, a superscript or a subscript with text
only as <hi rend='sub'><hi rend='bold'>foo</hi></hi>, or numbers with special
exponents, for instance 2<hi rend='sup'>nd</hi>, 3<hi rend='sup'>e</hi>.
```

In what follows, we consider only normal formulas. We have some difficulties with labels, tags and references. Consider for instance

```
\[ \tag{foo} x=y \label{a} \]  and $b=\eqref{a}$
```

Translation of the first formula is

```
<formula id='uid1' type='display'>
  <math mode='display' xmlns='http://www.w3.org/1998/Math/MathML'>
     <mrow>
       <mi>x</mi><mo>=</mo><mi>y</mi>
       <mspace width='2.em'/>
       <mo>(</mo><mi> foo </mi><mo>)</mo>
    </mrow>
  </math>
</formula>
```

The first important point is that the `\label` command adds an ID to the `<formula>` element, not to the math element or one of its sub-elements. As a consequence, it is impossible to use more than one label. Consider now the translation of the tag; MathML provides a strange method for equation numbering, that is badly interpreted by Firefox, and maybe other browsers. For this reason, it is not used by *Tralics*. There is a mechanism (described in full in the *Tralics* documentation) that allows you to change the behaviour of the `\tag` command. The default action is to put it at the end of the formula, with a bit of space and parentheses; multiple tags are merged, an optional star removes parentheses. It is possible to put the tag as an attribute to the formula, rather than putting it in the math. Note that a non-italic font will be used for the tag.

Translation of the second formula is

```
<formula type='inline'>
   <math xmlns='http://www.w3.org/1998/Math/MathML'>
     <mrow>
        <mi>b</mi><mo>=</mo><mo>(</mo><mref target='uid1'/><mo>)</mo>
     </mrow>
   </math>
</formula>
```

This was an error in previous version of *Tralics*. The same error was produced when putting the reference into a `\mbox`. The main reason is that MathML defines no `<mref>` element. In LaTeX, the reference is expected to print 'foo': note that the `\label` command remembers the value of the

`\tag`, it can be placed after the `\tag` command, this works since amsmath analyses the formula twice; the same method could be used by *Tralics*, since the label is interpreted (converted into an ID attribute) after translation of the formula. In our opinion, the correct method is to put the tag as an attribute to the formula, and this value should be used by the renderer for both formulas.

Consider now the following

```
95 \def\Big#1{{\hbox{$\left#1\vbox to11.5\p@{}\right.\n@space$}}}
96 $\Big($
```

This definition is taken from plain.tex, we shall discuss its use later. Let's ignore the last token, whose purpose is to make sure no additional horizontal space is added by this complicated construction. TEX creates a math formula; inside this formula, there is a `\hbox` that can contain any construction that can be inserted in a line of a paragraph (for instance an image); here it is a math formula; the math formula contains a `\vbox` (vertical stacking of items, normally space or horizontal boxes); here the box is empty, its width is zero, and its height is explicit. It is not possible in MathML to insert random elements in a math formula. Essentially, *Tralics* handles line 97 exactly as line 98:

```
97 $A\mbox{B$C$D F}E$
98 $A\text{B}C\text{D}\space\text{F}E$
```

We explained before that `\hbox`, `\mbox` and `\text` were parsed alike, with the exception that `\hbox` inserts the `\everyhbox` token list. Here, in line 98, we assume that the command `\text` contains only characters, maybe font changes, it will be translated into a `<mtext>` element. Since MathML provides no possibility of vertical stacking, the `\vbox` command produces an error. Note that `\hbox` and `\vbox` have special parsing rules in TEX, these are not implemented in math mode so do not try `\hbox to 2cm`.

7 The Code Generator

Let's start with the procedure that converts a table, because it is easy to explain. Consider the following example:

```
99  \begin{align}
100 \formulaattribute{tag}{8-2-3}
101 \thismathattribute{background}{white}
102 \rowattribute{mathvariant}{bold} x^2 + y^2+100 &=  z^2 \\
103 \multicolumn{1}{l}{\text{and}}\\
104 \cellattribute{columnalign}{left}  x^3 + y^3+1 &<  z^3
105 \end{align}
```

In some cases, the environment starts with an optional argument that specifies vertical alignment; it is followed by a mandatory argument that specifies horizontal cell alignment (non-characters are forbidden, characters other than r, l, c are ignored); for some environments like align, this argument must not be given (alignment here is rl repeated 5 times). The second argument of `\multicolumn` must be one of r, l or c; the effect is to add an attribute the current cell. The first argument of `\multicolumn` must be an integer k, if the current cell has number n, the next one has number $n + k$, and this number is used for determining its alignment; the number k is added as an attribute to the cell. The last argument of `\multicomumn` is the content of the cell.

Translation of the environment is in general a `<mtable>`, but fences may be added (for instance in the case of a matrix). Sometimes a displaystyle attribute is added to the table (for instance in the case of the '`align`' environment). In this case the current style is changed to displaystyle. In the case of the '`multline`' environment, the first and last cells have special alignment. If the array is terminated by a double-backslash, this produces a row with an empty cell. Hence: if the last cell of the array is empty it will be discarded, and if the last row is empty, it will be discarded too.

The example above contains some commands with name terminated by 'attribute'. The purpose is to add an attribute to (in order) the current formula, the current math element, the current row, and the current cell. In the example, the first cell in the last row has two attributes: its alignment is right (as defined by the table header) and left (as defined by the command). These attributes are set in the correct order, so that final alignment is left. The content of a cell is converted, as any other math list, according to the algorithm described below.

The code generator takes a sequence of tokens; it converts each token into a MathML object. In a second step, some special action in done if `\Big` commands are seen. In a third step, scripts are attached to kernels; in a fourth step, some `<mrow>` elements are added in order to get a correct size for delimiters. Finally, the resulting list is converted into a XML element. Let's start with this final step: normally all elements of the list are put in a `<mrow>` element (see note 5). If an explicit style command is seen, a `<mstyle>` element will be added.

The first step is defined by the following rules

- Space is ignored.
- Commands like `\hspace` read a value and produce math space.
- Commands like `\mbox`, `\ref`, are processed as expected (see above). They may produce `<mtext>` or math space elements.
- Commands like `\textstyle` change the current style.
- Command `\nonscript` is discarded in non-script style.
- Commands `\mathbin` and friends are interpreted (see below).
- Normally, a sequence of digits is converted into a `<mn>` element, and a sequence of characters into a `<mi>` element (see Note 1 above).
- Hat and underscore, `\right`, etc, are left unchanged. Note that the token or token list that follows hat or underscore is processed in a smaller style.
- Constants like `\alpha` are replaced by values found in tables.
- Lists are interpreted (recursion). In the case of `\left`, `\right`, delimiters are added. Arrays are handled here.
- All remaining tokens should be commands that take arguments. These arguments are processed, and something is done to them, but there are subtleties.
- In the case of `\mathchoice`, only one of the 4 arguments is used.
- In the case of `\operatorname*`, the argument should be a character string, and the result is a `<mo>`, classified as operator with nolimits or displaylimits. Same idea for `\qopname`.
- Commands like `\cellattribute` read their arguments, and install an attribute pair if possible, see example above.
- Commands like `\mathmi`, `\boxed`, `\smash`, `\phantom` are handled.
- Commands that generate fractions are considered here. We have a lot a variants, that may change the style of numerator, its horizontal alignment, the width of the rule, and delimiters may be added. Commands like `\xleftarrow` behave like a fraction, with rule replaced by an arrow. Commands that add accents are handled here. The result can be any XML object, in some cases it is flagged as a math operator, in some cases as a big object.

We consider now phase three. The current math list contains some XML elements, and some unprocessed TEX tokens, that are evaluated now. We consider three objects K, E and I (kernel, exponent, index), the idea is to add exponent and index to the kernel; there is a state variable, that can be looking for K or found K. A type T, an integer S, are also considered.

- If underscore or hat is seen while looking for K, an empty group `<mrow/>` is used for K, and K is found.
- If `\nonscript` is seen, the token that follows is discarded, provided it is an XML element of type space.
- If a command of type `\mathop` is found while looking for K, this defines the type T.
- If another command if found when looking for a kernel, this is an error. Otherwise, we have our kernel. If T is not defined by the previous rule, then T is the type of K.
- Consider now the case where we have T. If the current token is `\limits` or friends, this changes the behaviour of limits placement. If the command is `\displaylimits`, it will be replaced by `\limits` or `\nolimits` depending on whether or not current mode is display. This defines integer S.
- If the command is underscore or hat, then the object that follows becomes I or E. It is an error if the command is the last token in the list, if the object that follows is hat or underscore, if it not an XML object, or if the correspondin script is already given.
- If the previous rule does not apply (maybe because we are at the end of the token list), scripts are attached to the kernel, and parsing continues in mode looking for K.

Attaching an index is trivial: the result is a `<msub>` element with two children K and I. We have four cases to consider, because both I and E, or none, or one of both can be present. Now, MathML provides `<munder>`, this is the construct to be used if the index should be placed under the kernel (for instance a sum in display mode). The `<msub>` operator is the correct one if S says nolimits. It is also correct if T says nolimits (or displaylimits in non-display mode). Otherwise, the other operator must be used. If we use `<munder>` for a sum in non-display mode, this is incorrect, because MathML assumes by default that the operator has displaylimits as property (i.e., the movablelimits attribute is true). In some cases, *Tralics* sets it correctly to false, and sometimes it fails. If anything is attached to the kernel K, its type becomes big, otherwise it remains T.

Phase four of the algorithm is assumed to insert some fences (`<mrow>` elements) in order to control the size of delimiters like parentheses; it uses the fact that some elements are big, either a fraction, an array, an element with an accent, or a kernel with a script; it also uses the fact that some objects are of type variable-size. In a case like $(2x)^{-1}$, we have a variable size opening parenthesis, two small objects and a big one, the closing parenthesis with its superscript. This must be considered a closing delimiter (if we want to pair it) and a big object (our algorithm does nothing is nothing is big). Adding fences means to convert the formula into one of the following variants

```
106 $ { ( 2x ) ^{-1} }$
107 $ \left( 2x \right) ^{-1}  $
108 $ { ( 2x ) } ^{-1} $
```

The first variant is not the right one; other two variants are equivalent, and the last one is chosen. Note that the exponent must be attached to the object produced by the group (either `<mfenced>` or `<mrow>`). For this reason, if the kernel K is a variable size delimiter and has scripts, we add to the resulting list the kernel K and a special marker before K-with-scripts. At the end, the list is considered again; whenever we have a special marker, preceded by L, followed by K-with-scripts we replace the kernel of K-with-scripts by L, discard L and the special marker. In the case of the expression above, the algorithm constructs a group, and L is this group.

8 Adding Fences

Let's consider the following math expression

$$\int_0^\infty f(x+y)dx = |z| \ .$$

This is the same expression as shown in the introduction, without the `\mathord`. It is typeset by TEX in exactly the same way, but interpreted differently by *Tralics*. The reason is that the expression contains four stretchy operators, two parentheses and two vertical bars. There is no problem with operators that stretch horizontally (for instance text over an arrow or arrow over text), but something must be done with operators that stretch vertically; these are known to *Tralics* because they are of type Open, Close or Between (the command `\mathbetween` has been added to *Tralics*, it says that the element that follows should behave like a vertical bar).

In TEX, there is only one way to get a variable size object: it must be a delimiter used in a left-right scope, the same idea is used in MathML. In TEX a delimiter is a character that has a `\delcode`, a property that explains how to get larger versions of the object, in MathML it is an operator that has the stretchy property and can stretch. When you say `\left[` followed by some material and `\right]`, this defines a scope with two delimiters and the height and depth of the delimiters is the height and depth of everything else in the scope, the `\middle` command takes a delimiter as argument, and can be used in a left-right group.

In MathML there is no difference between a `<mfenced>` element containing A, bar, B, with parentheses as fences, and a `<mrow>` element that contains an open parenthesis, A, bar, B and the second parenthesis[4]. All three delimiters do stretch. Moreover, parentheses do stretch even if they are not the first or last item in the list. For some browsers, the size is not the same when the delimiter is the first or second item in the list. Currently, *Tralics* ignores the fact that some binary operators (slash for instance) can be stretchy.

Our algorithm handles left-right pairs in step one, this never causes problem. The translation of `\Big[` could be a left bracket that stretches at least and at most 120 percent of the size of the character: this possibility does not exists in TEX, and may be implemented in a future version of *Tralics*. The plain TEX definition of `\Big` is is

```
109 \def\Big#1{{\hbox{$\left#1\vbox to11.5\p@{}\right.\n@space$}}}
```

This code assumes that a ten point font is used. The amsmath package assumes on the other hand that `\big@size` contains the size of a normal delimiter, and scales it like this

```
110 \renewcommand{\Big}{\bBigg@{1.5}}
111 \def\bBigg@#1#2{%
112   {\@mathmeasure\z@{\nulldelimiterspace\z@}%
113      {\left#2\vcenter to#1\big@size{}\right.}%
114    \box\z@}}
```

Note that `\vcenter` is used instead of `\vbox`, but the definition is otherwise the same: we have a group, a box inside the group, and a math formula inside the box. A simpler definition could be

```
115 \def\Big#1{\left#1\vbox to11.5\p@{}\right.}
```

It is not possible to put a v-box in a MathML formula, but a phantom could be used instead.

The current algorithm (step two) is the following. When *Tralics* sees `\big` and friends, followed by a token T, it ignores the prefix, unless `\left T` is valid. The result will be of type left, right or middle, in case of `\bigl`, `\bigr` or `\bigm`. It will be of type left or right if T is an opening or closing delimiter. It will be of type middle otherwise. There is no difference between `\big`, `\Big`,

[4] The difference can be in the presence of a 'fence' attribute, that has, according to the MathML recommendation, no effect in the suggested visual rendering rules.

`\bigg` and `\Bigg`. After that, big-left and big-right are converted to `\left` and `\right` if properly nested (this implies that, in some cases, the prefix is ignored). For instance

```
$a\big(b\big)c$
```

is translated as

```
<mrow>
  <mi>a</mi>
  <mfenced open='(' close=')'><mi>b</mi></mfenced>
  <mi>c</mi>
</mrow>
```

Consider now step four of the algorithm. Each token in the list has a type, and according to these type, some operators are converted into fences. The type of `\Bigr[` is large right delimiter (this type is useful only in step two), the type of `\sum` is math operator with movable limits (this type explains how scripts should be attached to it). In step four, only six types are considered. First, we have types l, r, and m, that correspond to left, right and middle delimiters (these are the stretchy operators), then we have binary and relation operators (denoted by B and R). All remaining objects are classified as big or small; small objects are ignored. Big objects are fractions, symbols with scripts, etc. The type of an object is often correctly defined by default, but you can change it by adding a prefix. Example

```
$\mathopen a \mathclose b \mathord c \mathbin d \mathrel e\mathbetween  f $
```

A non trivial question is how to represent the type of a stretchy operator. For instance, in TEX, a vertical bar is of type ordinary, and preceded by `\bigm` it becomes a relation. For this reason a new type has been introduced, as well as the command `\mathbetween`.

Our rules will be explained on two examples, formulas (1) above and (2) here

$$\||f|^2 - |\frac{p_n}{q_n}|^2\|_{L^\infty(T)} < \varepsilon, \tag{2}$$

Line 16, as well as lines 123-124, show the sequence of tokens inserted in the math lists, while lines 17, 125, 127, and 129 show types constructed by Rule 1.

```
$$\mathopen\|\mathopen|f\mathclose|^2 - \mathopen|\frac{p_n}{q_n}
 \mathclose|^2 \mathclose\|_{L^\infty(T)} <\varepsilon,$$
MF: After find paren0 0b 1l 3r 4b
MF: matched 1, 3
MF: After find paren0 0l 1l 3r 5b 6B 7l 8b 9r 11b 12r 14b 18b
MF: rec 1, 3; 7, 9;
MF: After find paren0 0l 3b 4B 5b 7b 8r 10b 14b
MF: matched 0, 8
```

Rule 1. There is nothing to do if the formula has no big object, or no delimiters. In a case like $a(b)c$ we do nothing if the only big object is b. In a case like $a(b)c(d)e$ something is done if any element is big. More precisely, if we define the brace level of a token as the number of opening delimiters minus the number of closing ones before it (this can be a negative number) action is done if there is a token a brace-level zero or negative, or if two opening delimiters have been seen. In case something is done, the type of useful tokens is printed (the last item on the line is the length of the list).

Our algorithm is recursive. Line 125 corresponds to the index attached to the double bar. This double bar is item number 12 in the list. Since it is preceded by the command `\mathclose`, it is a closing delimiter, hence you see 12r. Item 13 is a special marker, and item 14 is the K-with-scripts (double bar with subscript) that must be merged with item 12.

Rule 2. Fences are recursively added if possible and useful. In a case like $([a] = |c|) + [b|c] = (e|f|g)$ fences are added around brackets, because they match, and contain at most one middle

delimiter. The first pair of parentheses does not match, because they are not at outer level, the second pair because there are two vertical bars. Fences are added for all matching pairs, if at least one is not at outer level[5]. Example: see line 128. The algorithm restarts with rule 1. The remaining type list is shown on line 129. Remaining delimiters are properly matched, but there is no nesting, and rule 3 will be used.

Rule 3. In a case like $(x^2)+(y^2|z)$, that contains alternatively opening and closing delimiters, and at most one middle delimiter in each group, obvious fences are used, and that's all. The trace will contain a line of the form

```
131 MF: matched 0, 2  4, 8
```

This means that the first fence starts with token 0 and ends with token 2, the second fence with token 0 and ends with token 2. See also line 130.

Rule 4. In a case like $\int |x|$, we split the formula in two parts: before and after the big. What precedes the big is handled by the rule 5, after that, rule 2 is applied, except that if what follows the big contains only delimiters (of whatever type) obvious fences are used. The trace will show

```
132 MF: LBR 1 3
```

Rule 5. The algorithm considers sublists (see line 18). In example 1, everything after the integral sign is considered. This list is further divided according to binary or relation operators. For instance $|a+b|=|c+d|$ can be split at the equals sign, but not $a+|b=c|+d$, and $f(x)+g(y)$ can be split at the plus sign. We check two cases: binary and relation, or relation. The example on line 18 shows that the plus at 4B is discarded. It shows on lines 19 and 22 the two sublists (lines 20 and 23 are the same, where numbers in parentheses indicate position of first and last element in the sublist). In some cases pairing is OK, and you will see a line like 21, where fences are inserted at the location of the delimiter, and sometimes pairing fails, and fences are added at the start or end of the subformula (see line 24).

Conclusion: the algorithm described here is implemented in *Tralics* version 2.10.9; it has been tested on all examples known to the author. Cases where the algorithm fails exist and you can submit a bug report to the author; he will use them to improve the quality of the software.

References

[1] Thierry Bouche. A pdflatex-based automated journal production system. *TUGboat*, 27(1), 2006.

[2] David Carlisle, Michel Goossens, and Sebastian Rahtz. De XML à PDF avec `xmltex` et PasiveTEX. In *Cahiers Gutenberg*, number 35-36, pages 79–114, 2000.

[3] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier (editors). Mathematical Markup Language (MathML) Version 2.0, 2001.

[4] José Grimm. Tralics, a LaTeX to XML translator, Part I. Rapport Technique 309, Inria, 2006.

[5] José Grimm. Tralics, a LaTeX to XML translator, Part II. Rapport Technique 310, Inria, 2006.

[6] José Grimm. Producing MathML with Tralics. Rapport de Recherche 6181, Inria, 2007.

[7] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[5] This additional subclause may be relaxed, because rendering of formulas (1) change if we swap LHS and RHS. Remedy is unclear.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Futurs : Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399